Static Mapping of Subtasks in a Heterogeneous Ad Hoc Grid Environment

Sameer Shivle[1], Ralph Castain[1], H. J. Siegel[1,2], Anthony A. Maciejewski[1], Tarun Banka[2], Kiran Chindam[1], Steve Dussinger[1,3], Prakash Pichumani[1], Praveen Satyasekaran[1], William Saylor[1], David Sendek[1], J. Sousa[1,3], Jayashree Sridharan[1], Prasanna Sugavanam[1], and Jose Velazco[4]

Colorado State University
[1]Electrical and Computer Engineering Dept.
[2]Computer Science Dept.
Fort Collins, CO 80523, U.S.A
{ssameer, hj, aam, kiran, prkash, moses, jaya, prasanna}@engr.colostate.edu
{rcastain, sendekdm}@lamar.colostate.edu
tarunb@cs.colostate.edu

[3]HP Technologies
Fort Collins, CO 80528-9544
{sjd, jso}@fc.hp.com

[4]Abbott Laboratories
Barceloneta, Puerto Rico 00617
jose.velazco@abbott.com

Abstract

An ad hoc grid is a heterogeneous computing and communication system without a fixed infrastructure; all of its components are mobile. Energy management is a major concern in an ad hoc grid. One important aspect of energy management is to minimize the energy consumption during a mission. In an ad hoc grid, communication and computations are deeply intertwined, and any energy optimization must consider both types of activities together rather than separately. The mapping (defined as matching and scheduling) of tasks onto machines with varied computational capabilities has been shown, in general, to be an NP-complete problem. Therefore, heuristic techniques are required to efficiently map tasks to machines in an ad hoc grid so as to minimize the energy consumed due to communication and computation. This research evaluates and compares energy management issues for resource allocation in ad hoc grids using six static heuristics.

1. Introduction and Problem Statement

An ad hoc grid is a heterogeneous computing and communication system without a fixed infrastructure (i.e., all of its components are mobile). An ad hoc grid allows a group of individuals to accomplish a mission that involves extensive computation and communication among the grid components, often in a hostile environment; examples of applications of ad hoc grids are disaster management, wildfire prevention, and peacekeeping operations [18]. In all these cases a grid-like environment is necessary to reliably support the coordinated effort of a group of individuals working under extreme conditions. As the total battery energy available for any of the above applications is limited, energy management is a major concern in ad hoc grids. Thus, it becomes necessary to allocate efficiently the application task to the resources in the grid to minimize the total battery usage. This study focuses on this aspect of resource allocation in an ad hoc grid, where the primary objective is to minimize the total battery energy used to successfully accomplish a mission.

For this research, a single, large application task is considered to be composed of S communicating subtasks with data dependencies among them. This application task is to be executed in an ad hoc grid as part of the mission being conducted.

An important research problem is how to assign resources to the subtasks (matching) and order the execution of the subtasks that are matched (scheduling) to maximize some performance criterion of a

This research was supported in part by the Colorado State University George T. Abell Endowment.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

heterogeneous computing (HC) system. This procedure of matching and scheduling is called mapping or resource allocation. Static mapping refers to the case where the applications are mapped in an off-line planning phase [6], e.g., in a production environment. The mapping problem has been shown, in general, to be NP-complete (e.g., [8, 10, 12]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 3, 4, 5, 6, 9, 11, 17, 19, 23]).

In this study, we statically (offline) find a resource allocation for a single application task composed of communicating subtasks that will be needed for a mission to be completed. The idea is to come up with a complete static mapping of subtasks to machines that could be used later when a particular mission is to be instantiated (e.g., response to a particular wildfire). The application would use input data relevant to the particular mission that is instantiated. The goal is to map subtasks to machines in such a way as to minimize the average percentage of the energy consumed by a machine.

Six static mapping heuristics for this problem are evaluated and compared in this study through simulation experiments. The simulated HC environment consists of M machines in the ad hoc grid. The estimated expected execution time for each subtask on each machine is assumed to be known *a priori*. The estimated time to compute (ETC) values (calculated using the gamma distribution method in [2]) are used by the mapping heuristics. The estimated execution time of subtask i on machine j is $ETC(i, j)$. Each machine j has four energy parameters associated with it:

i. maximum battery energy: $B(j)$;
ii. rate at which it consumes energy for subtask execution, per ETC time unit: $E(j)$;
iii. rate at which it consumes energy for subtask communication, per communication time unit: $C(j)$; and
iv. the machine's communication bandwidth: $BW(j)$.

Parameters (ii) and (iii) use a simplified model of real energy consumption.

The energy consumed for executing a single subtask i on machine j is $ETC(i , j) \times E(j)$. The time required to transfer one bit of a data item between machine j and machine k is the inter-machine communication time called $CMT(j, k)$ and is given by:

$$CMT(j,k) = 1/\min\left(BW(j), BW(k)\right).$$

The energy consumed to send a data item g of size $|g|$ from machine j to machine k is $CMT(j, k) \times C(j) \times |g|$. Each machine can transmit data to only one destination at a time, and can do so while it is computing. A machine can simultaneously handle one outgoing data transmission and one incoming data reception. Similar to the study in [22], we assume that:

i. a subtask can send out data only after it has completed execution; and
ii. a subtask may not begin execution until it receives all of its input data items.

The ad hoc grid that is considered for this project is a simplified version of an actual one. The list of simplifying assumptions that have been made are as follows:

- the energy consumed by a subtask to receive a data item is ignored;
- any initial data (i.e., data not generated during execution of the application task) is preloaded before the actual execution of the application task begins;
- a machine consumes no energy if it is idle (i.e., not computing or not transmitting).

The performance metric is based on the energy consumption across all the machines in the ad hoc grid. The total battery energy consumed by a machine j after the entire task has been completed is given by $EC(j)$. The performance metric, B_{pavg} used to evaluate the mapping is defined as the percentage of energy consumed by each machine to complete the entire task, averaged across all machines, and is given by

$$B_{pavg} = \frac{\sum_{j=0}^{|M|-1} \left(EC(j)/B(j)\right)}{|M|}.$$

The goal of this study is to map all the subtasks to machines in such a way as to minimize B_{pavg}, while meeting an application execution time constraint τ. Six static mapping schemes are studied in this paper: Min-Min, Levelized Weight Tuning, Genetic Algorithm, Simplified Lagrangian, Bottoms Up, and A*. The makespan is defined as the overall execution time of the application task on the machine suite in the ad hoc grid. So the final makespan of all mappings has to be less than or equal to τ. The wall clock time for each mapper itself to execute is required to be less than or equal to 60 minutes on a typical unloaded 1 GHz desktop machine.

The next section describes the simulation setup used for this research. Section 3 provides a list of some of the literature related to this work. In Section 4, the heuristics studied in this research are presented. Section 5 describes the results, and the last section gives a brief summary of this research.

2. Simulation Setup

In this study, the application task is composed of 1024 communicating subtasks. This large number of subtasks is chosen to present a significant mapping challenge for each heuristic.

The data dependencies among the subtasks are represented by a directed acyclic graph (DAG). The pseudocode to generate the DAG is given in the appendix of this paper. For this study, ten different DAGs are developed. The maximum fan-in and fan-out values for all the ten DAGs generated are twelve and two, respectively. Also, for each DAG there are seven subtasks with no predecessors, seven subtasks with no successors, and the remaining 1010 subtasks have predecessors and successors. The sizes of the global data items to be transferred from one subtask to another are sampled from a Gamma distribution, with a mean value of 2.8 megabits and a variance of 1.4 megabits.

There are a total of eight machines in the simulated ad hoc grid, and these are divided equally into two classes: "fast machines" and "slow machines." The ETC matrices are setup such that machines 0 to 3 are fast machines, while machines 4 to 7 are slow machines. There are eight communication channels available that allow the eight machines to communicate simultaneously with each other.

The ETC values for all subtasks, taking heterogeneity into consideration, are generated using the Gamma distribution method described in [2]. For this research, a task mean and coefficient of variation (COV) are used to generate the ETC matrices. The mean subtask execution time is chosen to be 100 seconds and a COV of 0.9 is used to generate an ETC matrix with high task and high machine heterogeneity. For this study, ten different ETC matrices are generated.

To obtain the two classes of machines, all the ETC values for the slow machines are adjusted by a multiplicative factor (*MF*). For each subtask i the ratio $diff_i$, of the ETC value of the fastest slow machine to the ETC value of the slowest fast machine is calculated as

$$diff_i = \left(\frac{\min ETC(i, j) \text{ for } j \in [4,7]}{\max ETC(i, j) \text{ for } j \in [0,3]} \right).$$

Then the value of MF is given by

$MF = 2 / (\min diff_i \text{ for } i \in [0, 1023])$.

All the ETC values for the slow machines are now multiplied by the MF to get the new adjusted values. After creating the two classes of machines, the new mean estimated execution time for a single subtask is 131 seconds. For this study, across all the subtasks in an ETC matrix, the average fastest machine is approximately ten times faster than the average slowest machine.

	fast machines	slow machines
$B(j)$	580 energy units	58 energy units
$C(j)$	0.2 energy units/ sec	0.002 energy units/sec
$E(j)$	0.1 energy units/ sec	0.001 energy units/sec
$BW(j)$	8 megabits/sec	4 megabits/sec

Table 1: The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for fast and slow machines.

The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for both fast and slow machines are shown in Table 1. These values represent a rough industry average based on microprocessors and battery capacity selected on currently commercially available machines. Fast machines are typified by the DELL Precision M60 notebook computer using an Intel MP4M processor operating at 1.7GHz. The statistics for the slow machines are typical personal digital assistant (PDA) computers, such as the DELL Axim X5 that uses an Intel PXA255 processor operating at 400 MHz.

The value of the time constraint τ is chosen so that it prevents any heuristic from mapping subtasks only to slow machines, which consume less energy to execute a subtask. A simple greedy mapping heuristic was used to determine the value of τ as 34075 seconds. The performance of each heuristic is studied across 100 different scenarios, where each scenario is a combination of one of the task graphs and one of the ETC matrices.

3. Related Work

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. The nature of the DAGs used in this study are similar to those used in [22]. Similar to [22], this study also has a single application that is decomposed into a number of communicating subtasks with data dependencies among them, represented by a directed acyclic graph.

Three of the six heuristics presented in this paper, namely Min-Min, Genetic Algorithm, and A*, have been used previously to map tasks onto heterogeneous machines (e.g., [6]). However unlike [6], where the goal was to minimize the total time required to complete an application task, the goal of our study is to minimize the average percentage of energy consumed by the machines, in addition to complete the entire task

within a time constraint. The Min-Min heuristic has proven to be a good heuristic for dynamic and static mapping problems in earlier studies (e.g., [6, 17]). The Bottoms Up heuristic used in this study is a variation of the Min-Min heuristic. Bottoms Up assigns tasks to machines in a manner similar to the Min-Min heuristic, but considers tasks for scheduling in a different manner. Genetic Algorithms are a technique used for searching large solution spaces and have been used for mapping subtasks to machines in a HC environment (e.g., [6, 21, 22]). The Genetic Algorithm used in this study is a slightly modified version of the one used in [22]. A* is a search technique based on a tree and has been used for many task allocation problems (e.g., [6, 15, 7]). A* has been selected for this study because the application task is represented in the form of a DAG, and A* has been found to be highly effective in searching a tree or graph [15]. The Simplified Lagrangian heuristic presented in this paper is a modified version of the one used in [16]. Lagrangian relaxation techniques have been used in [16] for job scheduling in an industrial environment.

4. Heuristics

For all the six heuristics except Bottoms Up, only the subtasks whose predecessors had been fully mapped could be considered during a given mapping iteration (referred to as mappable subtasks). Also, for the final mapping of all the six heuristics, the energy constraint is that $B(j)$ is not exceeded for any machine, and the time constraint is that the execution time of the application does not exceed τ. This section describes the six heuristics and a lower bound on the objective function.

4.1. Min-Min

Based on the Min-Min concept in [12], this heuristic utilized a fitness function to evaluate all mappable subtasks. The fitness function is chosen such that it would reflect the change in B_{pavg} and also the change in the makespan of the system if a subtask is mapped on to a machine. Let $PB_{pavg}(i, j)$ be the partial B_{pavg} of the system, and let $PCT(i, j)$ be the partial completion time of machine j normalized with respect to τ, if subtask i was mapped to machine j. Then using α as a weighting parameter, the fitness value $f(i, j)$ of any subtask i on machine j is calculated as:

$$f(i, j) = \alpha \times PB_{pavg}(i, j) + ((1 - \alpha) \times PCT(i, j)).$$

The Min-Min heuristic can be summarized by the following procedure.

1. A list of mappable subtasks is created. Initially this list consists of subtasks with no predecessors.
2. For each subtask i in the above list, across all machines find the machine j that gives the subtask its minimum fitness value $f(i, j)$, ignoring other subtasks in the list. This is the first "Min."
3. From among all the subtask/machine pairs found in step 2, find the pair that gives the minimum fitness value. This is the second "Min."
4. The subtask found in the above step is then removed from the list of mappable subtasks and is mapped to its paired machine.
5. Update the time and energy availability of the machine on which the subtask is mapped and also across all machines that send global data items to the mapped subtask.
6. The set of mappable subtasks is updated to include any other new subtasks whose precedence constraints have now been met.
7. Repeat steps 2 to 6 until all the subtasks are mapped and calculate the value of B_{pavg}.

The procedure from step 1 to step 7 is carried out for eleven different values of the weighting factor α to get eleven different mappings. The value of α was varied from 0 to 1 in steps of 0.1. From among the eleven different mappings, the mapping that gave the smallest value of B_{pavg} and also met the energy and time constraints is chosen as the final mapping.

4.2. Levelized Weight Tuning

In a manner similar to that used in [13] and as shown in Figure 1, the Levelized Weight Tuning (LWT) heuristic assigns subtasks to different levels depending on the data precedence constraints.

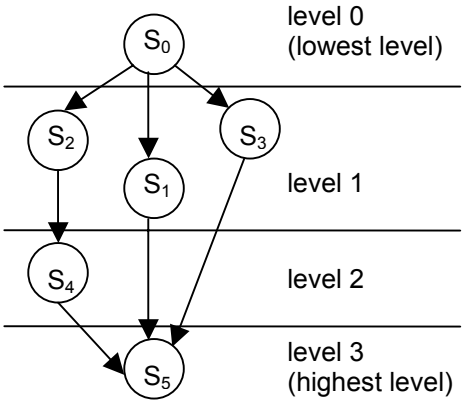

Figure 1: Levelizing of subtasks S_0, S_1, S_2, S_3, S_4, and S_5 for a given sample DAG.

The lowest level consists of subtasks with no predecessors and the highest level consists of subtasks

with no successors. Each of the rest of the subtasks is at one level below the lowest producer of its global data items. Starting from the lowest level, each subtask on its respective level is assigned a priority based on the total size (sum) of its output global data items. The subtasks with larger sums of output global data items have a higher priority on their respective level.

The LWT heuristic can be summarized by the following procedure.

1. All the subtasks are first assigned levels depending on the precedence constraints. Subtasks on each level are assigned a priority as described above.
2. Starting from the lowest to the highest level, subtasks are considered for mapping by levels (from low level to high level), and by priorities (from high priority to low priority) within levels.
3. Every time a subtask is considered for mapping, find a machine M1 that will increase the current B_{pavg} of the system by the least percentage. Let this least percentage increase in B_{pavg} be $MinB_{pavg}$. Also, find a machine M2 that will increase the current makespan of the system by the least percentage. Let this least percentage increase in makespan be *MinMspan*. A threshold factor η, which is the ratio of makespan to τ is calculated.
4. If $\eta > 0.9$,
   then the subtask is mapped to machine M2,
   else
   if $((1 - B_{pavg}) \times MinB_{pavg}) > (B_{pavg} \times MinMspan)$
   then the subtask is mapped to machine M1
   else, the subtask is mapped to machine M2.
5. Update the time and energy availability of the machine on which the subtask is mapped and also across all machines that send global data items to the mapped subtask.
6. Repeat steps 2 to 5 until all the subtasks are mapped and calculate the final value of B_{pavg}.

4.3. Bottoms Up

The Bottoms Up (BU) heuristic assigns subtasks to levels in a manner similar to the LWT heuristic. However, unlike LWT, the BU heuristic begins by mapping subtasks from the highest level. Thus, for the BU heuristic, the set of mappable subtasks at any given time consists of all subtasks that either have no successors or subtasks whose successors have previously been mapped. Subtasks in each level are randomly selected for mapping.

Let the time for execution and communication of subtask i on machine j, normalized with respect to the maximum time required for execution and communication by subtask i across all machines be $NT(i, j)$. Let the energy consumed for execution and output communication of subtask i on machine j, normalized with respect to the maximum energy consumed for execution and output communication of subtask i across all machines, be $NE(i, j)$. Then, using β as a weighting parameter, the fitness value γ_{ij} is calculated as

$\gamma_{ij} = (\beta \times NT(i, j)) + ((1 - \beta) \times NE(i, j))$.

Different values of the weighting factor β were considered for this study. The weighting factor of $\beta = 0.52$ was found to give the best value of B_{pavg} within the time and energy constraints for all the scenarios and hence was selected for this study.

The BU heuristic can be summarized by the following procedure.

1. All the subtasks are first assigned levels depending on the precedence constraints as explained above.
2. Starting from the highest level to the lowest level, all mappable subtasks are considered randomly for mapping within the respective level
3. For each mappable subtask i, at the current level and across all machines find the machine j that gives the subtask its minimum fitness value γ_{ij}, ignoring other subtasks on that level.
4. From among all the subtask/machine pairs found in the above step, find the pair that gives the minimum fitness value.
5. The subtask found in the above step is then assigned to its paired machine.
6. Repeat steps 2 to 5 for each level (from highest to lowest level) until all subtasks are assigned machines.
7. After all subtasks are assigned machines, they are scheduled in the reverse order they were matched.
8. The entire mapping is then evaluated and the final value of B_{pavg} is calculated.

4.4. A*

The A* technique used in this study is similar to that used in [6, 7]. A* is a tree-search algorithm, beginning at a root node that is a null solution. As the tree grows, nodes represent partial mappings (a subset of subtasks is assigned to machines). The partial mapping (solution) of a child node has one more subtask mapped than the parent node. For each node n, a cost function $c(n)$ is calculated as follows:

Let $g(n)$ be the maximum of the machine completion times for the subtasks mapped through node n and $h(n)$ be a lower bound estimate of the completion time of all the unassigned subtasks at node n. Let $mmct(n)$ be the maximum of the minimum machine completion times over all unassigned subtasks at node n. Then the function $h(n)$ is defined as

$h(n) = \max\left(0, \left(mmct(n) - g(n)\right)\right)$.

The function $f(n)$ that is an estimate of the time required to complete all the subtasks, normalized with respect to τ, is given by

$$f(n) = \big(g(n) + h(n)\big)/\tau .$$

The function $p(n)$ is the lower bound of the estimated energy consumption through node n. It is defined as the sum of the B_{pavg} of all the assigned subtasks at node n and the lower bound estimate of the B_{pavg} for all the unassigned subtasks at node n. The lower bound estimate of the B_{pavg} is calculated by assuming that every unassigned subtask is assigned to a machine that increases the B_{pavg} of the system by the least amount.

The cost function for node n is then given by $c(n) = \sqrt{\big(\mu \times f(n)^2\big) + p(n)^2}$, where μ is a weighting factor. Different values of the weighting factor μ were considered for this study. The weighting factor of μ = 0.07 was found to give the best value of B_{pavg} within the time and energy constraints for all the scenarios and hence was selected for this study.

The A* heuristic can be summarized by the following procedure.

1. A valid scheduling order of subtasks that satisfies the precedence constraints for the entire task is first generated.
2. All subtasks are then considered for mapping in the order that they are in this valid schedule.
3. The root node generates eight nodes (partial mappings) by allocating the first mappable subtask to each of the eight machines.
4. After a parent node generates child nodes, it becomes inactive (i.e., it is not eligible for further expansion). The new nodes created are considered to be active nodes and are stored in a node list. The size of the node list is always kept at 100 by retaining only the best 100 active nodes (based on $c(n)$) at any one time. Similar to [6], this is done to keep the execution time of the heuristic tractable.
5. For the next mappable subtask the node with the minimum $c(n)$ in the node list is then expanded to generate eight more new child nodes (corresponding to mapping that task to each of the eight machines).
6. Repeat steps 2 to 5 for every mappable subtask until finally a node is expanded to give, eight complete mappings. From these eight complete mappings, the mapping that gives the best value of B_{pavg} and also meets the energy and time constraints is then selected as the final mapping.

Experiments with node lists of sizes larger than 100 were also conducted. However, it was found that there was no significant improvement in the value of B_{pavg}, but the heuristic execution time increased considerably.

4.5. Simplified Lagrangian

Lagrangian based approaches have been applied to solve a wide range of complex production scheduling problems [16]. The technique used here is a simplified version of [16] so that it would be suitable for the problem environment in this study. At any time k, if the energy remaining in machine j is denoted $ER(j, k)$ and the makespan is denoted $makespan(k)$, then the Lagrangian equation, $L(\delta, k)$ is given by

$$L(\delta, \mathrm{k}) = \delta\left(\sum_{j=0}^{|M|-1} ER(j,k) \Big/ |M|\right) + (1-\delta)\left(1 - \left[makespan(k)/\tau\right]\right).$$

Different values of the weighting factor δ were considered for this study. The weighting factor of δ = 0.8 was found to give the best value of B_{pavg} within the time and energy constraints for all the scenarios and hence was selected for this study.

The Simplified Lagrangian (SL) heuristic can be summarized by the following procedure.

1. At every mapping event, the next available machine (i.e., the machine with the minimum machine availability time) is selected. If one or more machines have the same minimum machine availability time, then any one of these machines is selected randomly.
2. For the selected machine, the list of mappable subtasks is generated. The list of mappable subtasks consists of all the subtasks whose predecessors have been mapped and can begin execution on the selected machine.
3. Find the potential contribution of each mappable subtask in the above list to the system Lagrangian (i.e., $L(\delta, k)$), ignoring other subtasks in the list.
4. From among the mappable subtasks found in the above step find the subtask that gives the largest value of the system Lagrangian, $L(\delta, k)$.
5. The subtask found in the above step is then removed from the list of mappable subtasks and is mapped to its selected machine.
6. Update the time and energy availability of the machine on which the subtask is mapped and also across all machines that send global data items to the mapped subtask.
7. Repeat steps 1 to 6 until all the subtasks are mapped and calculate the value of B_{pavg}.

The SL allowed a mappable subtask to be scheduled for a time prior to the target machine's availability time if a sufficiently large "hole" in the

existing schedule could be found that complied with precedence constraints. As a result, the SL-generated mappings exhibited a very small makespan as compared to all the other heuristics.

4.6. Genetic Algorithm

This method is similar to the genetic algorithm approach used in [22]. The genetic algorithm (GA) operates on a population of 100 chromosomes. Each chromosome represents one solution to the problem and a set of chromosomes is called a population. Each chromosome is made of a scheduling string and a matching string. The scheduling string is a total ordering of the subtasks in the DAG that obeys the precedence constraints, while the matching string gives the subtask-to-machine assignments. To form a scheduling string, the DAG is topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated (similar to the mutation procedure described below) a random number of times to generate 98 other valid scheduling strings. The corresponding 99 matching strings are generated by randomly assigning subtasks to machines. The population also includes one chromosome (seed) that is the Bottoms Up solution. Similar to the approach in [22], these chromosomes then undergo selection, crossover, mutation, and evaluation.

Each chromosome has a fitness value (B_{pavg}) associated with it. The rank-based roulette wheel scheme is used for selection [21]. This scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. *Elitism*, the property of guaranteeing the best solution remains in the population, is also implemented [20]. The population size stays fixed at 100.

In the crossover step, a pair of parent chromosomes is selected from the chromosome population. In case of scheduling string crossover, for each pair a random cut-off point that cuts the scheduling strings into top and bottom parts is generated. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings are valid scheduling strings. For matching string crossover, again a random cut-off point that cuts the matching strings into top and bottom parts is generated. Then the machine assignments of the subtasks in the bottom parts are exchanged. After the crossover operation for both the scheduling and the matching strings, the new chromosomes generated are evaluated and if the new chromosomes generated do not violate energy or time constraints, then they replace the parent chromosomes in the population; else the new chromosomes are dropped and no child chromosomes are created.

In the mutation step, a parent chromosome is selected for mutation from the chromosome population. In case of scheduling string mutation, for each chosen parent scheduling string, a subtask (called victim subtask) is selected randomly. This victim subtask is then moved randomly to another position in the scheduling string in such a way that it does not violate any precedence constraints to obtain a new valid scheduling string. In case of matching string mutation, for each chosen parent matching string, two subtask/machine pairs are selected randomly and their machine assignments are swapped. Similar to crossover, after the mutation operation for both the scheduling and matching strings, the new chromosomes generated are evaluated and if the new chromosomes generated do not violate energy or time constraints, then they replace the parent chromosomes in the population; else the new chromosomes are dropped and no child chromosomes are created.

For both crossover and mutation operation, the chromosome population is traversed serially, and every chromosome is considered for crossover with a probability of 40% and for mutation with a probability of 20%. Selection, crossover, mutation, and evaluation steps constitute a single GA iteration The GA stops after a total of 400 iterations. Until the stopping criterion is met, the loop repeats, beginning with the selection step. At the end of 400 iterations, the chromosome that gave the best B_{pavg} is selected as the final mapping. For this study, at any point of time only chromosomes that did not violate the energy or time constraint were allowed to be in the population and the population size was always kept constant at 100 chromosomes.

4.7. Lower Bound (LB)

The method developed for estimating a lower bound (LB) on B_{pavg} for this study ignores data precedence constraints, inter-machine communications, the battery power constraint, and τ. For each subtask (in any random order) in the application task, the minimum percentage energy it will consume over all the machines is found. These minimum percentage energy values for all the subtasks are summed up and then finally averaged over all machines. This gives us a LB on B_{pavg}. Thus, the LB can be given as

$$\frac{1}{|M|} \times \sum_{i=0}^{|S|-1} \left(\min \left(\frac{ETC(i,j) \times E(j)}{B(j)} \right) \quad for \;\; j \in [0,7] \right).$$

5. Results

The simulation results are shown in Figures 2, 3, and 4. All heuristics are run for 10 different task graphs (DAGs) and 10 different ETCs (i.e., a total of 100 combinations) and the average values and 95% confidence intervals [14] are plotted. The running times of the heuristics averaged over 100 trials, mapping 1024 subtasks per trial, are shown in Table 2.

heuristic	average execution times (seconds)
Min-Min	19
Levelized Weight Tuning	670
Bottoms Up	0.7
A*	645
Simplified Lagrangian	1200
Genetic Algorithm	3200

Table 2: The execution times of the heuristics averaged over 100 scenarios (using a typical 1 GHz unloaded machine).

Among the faster heuristics (i.e., Min-Min and Bottoms Up), the Bottoms Up heuristic did slightly better than the Min-Min heuristic and gave the best B_{pavg}. Both these heuristics are basically two-phase greedy heuristics that optimize a fitness function. The major difference between the two is that Min-Min used the top to bottom approach beginning from the root node to the leaf node of the subtask graph, whereas Bottoms Up used the bottom to up approach.

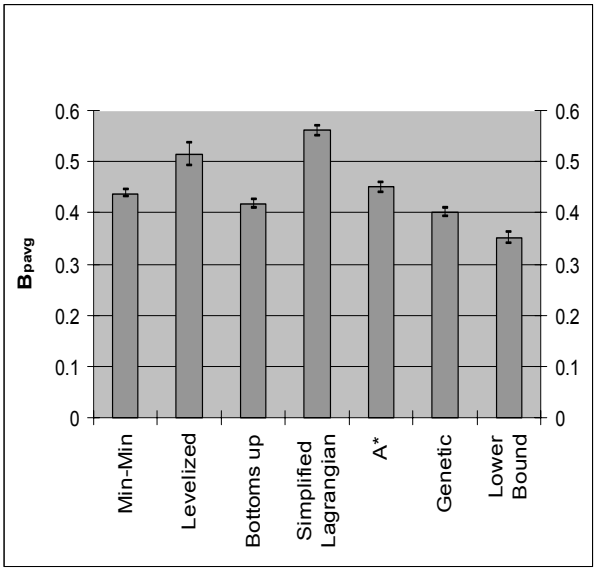

Figure 2: The simulation results for B_{pavg}.

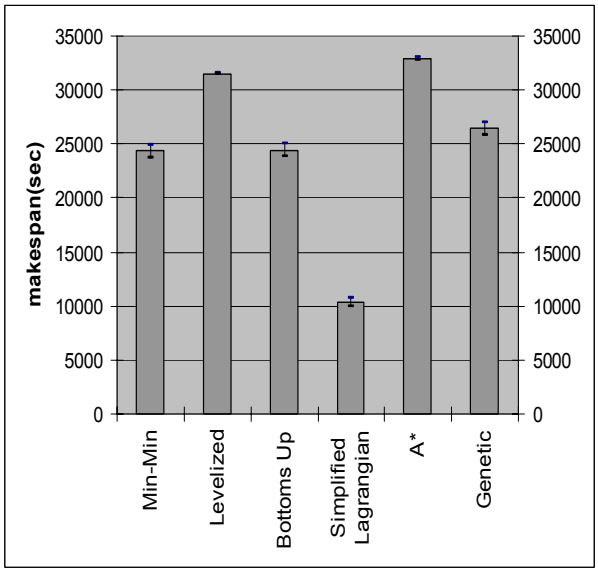

Figure 3: The simulation results for makespan.

Overall among all the heuristics, the Genetic Algorithm performed the best. This was expected because the GA used the Bottoms Up result as a seed and also because it used the concept of elitism that ensured that the B_{pavg} of the new solution obtained was either better or at least the same as the seed.

The Simplified Lagrangian had the highest average B_{pavg} because it tried to optimize the makespan along with the main objective function of B_{pavg}. It tried to fill in the gaps in the machine subtask queues when the machine was not computing and waiting for global data items, by allowing a mappable subtask to be scheduled for a time prior to the target machine's availability time if it was possible to do so without violating precedence constraints. As described below, this resulted in a higher average usage of fast machines, which in turn leads to a higher B_{pavg}. As seen in Figure 3, the makespan generated by the Simplified Lagrangian is significantly less than that of the other heuristics.

Another parameter, called packing density, was used to study the behavior of the heuristics for the given problem. Packing density is defined as the ratio of the total time spent by a machine for subtask execution only (ignoring the time required for communication) to the total makespan. As seen from Figure 4, the Simplified Lagrangian had a higher average packing density over all machines, especially the fast machines. Thus, for all the heuristics except the Simplified Lagrangian, the fast machines had many time gaps when the machines were not doing any computation but were waiting for global data items.

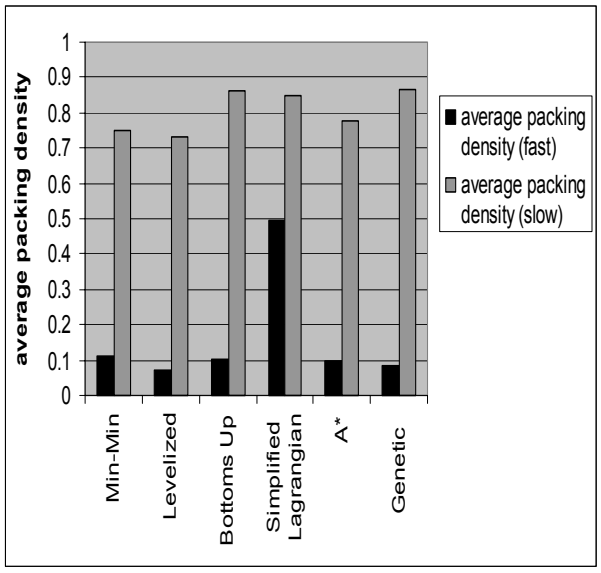

Figure 4: The simulation results for average packing density across fast machines and slow machines.

6. Summary

Six static heuristics were designed, developed, and simulated using the HC environment presented. Application tasks composed of communicating subtasks with data dependencies were mapped using the heuristics described in this research.

The best B_{pavg} value was obtained by using the Genetic Algorithm and the second best by using Bottoms Up. However, the Genetic Algorithm used Bottoms Up as a seed and on an average did only 3.9% better than Bottoms Up. Also, the time required for the Genetic Algorithm itself to execute (i.e., heuristic execution time) is extremely high as compared to the Bottoms Up heuristic. Thus, Bottoms Up seems to be a good choice for the given problem.

Acknowledgements: The authors thank Shoukat Ali and Jong-Kook Kim for their valuable comments.

References

[1] S. Ali, J. K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. Prasanna, "Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2002*), June 2002, pp. 519-530.

[2] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, Special 50th Anniversary Issue, Vol. 3, No. 3, Nov. 2000, pp. 195-207 (invited).

[3] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM "*Proceedings of the 15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*)," paper HCW 15, Apr. 2001.

[4] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM "*Proceedings of the 15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*)," paper HCW 06, Apr. 2001.

[5] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems," *2001 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2001*), June 2001, pp. 1–12 (invited keynote paper).

[6] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and Bin Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810-837.

[7] K. Chow and B. Liu, "On mapping signal processing algorithms to a heterogeneous multiprocessor system," *International Conference on Acoustics, Speech, and Signal Processing (ICASSP '91)*, Vol. 3, 1991, pp. 1585-1588.

[8] E. G. Coffman, Jr. ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[9] M. M. Eshaghian, ed., *Heterogeneous Computing*. Norwood, MA, Artech House, 1996.

[10] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[11] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, San Fransisco, CA, Morgan Kaufmann, 1999.

[12] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280-289.

[13] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th IEEE Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 93-100.

[14] R. Jain, "*The Art of Computer Systems Performance Analysis Techniques for Experimental Design,*

Measurement, Simulation, and Modeling," New York, Wiley, 1991.

[15] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No 3, July-Sep. 1998, pp. 42-51.

[16] P. Luh, X. Zhao, Y. Wang, and L. Thakur, "Lagrangian relaxation neural networks for job shop scheduling," *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 1, Feb. 2000, pp. 78-88.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107-121.

[18] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, and H. J. Siegel, "Ad hoc grids: Communication and computing in a power constrained environment," *Workshop on Energy-Efficient Wireless Communications and Networks 2003 (EWCN 2003),* cosponsors: IEEE Computer Society and IEEE Communications Society, *Proceedings of the 22nd International Performance, Computing, and Communications Conference (IPCCC)*, Apr. 2003.

[19] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, New York, NY, Springer-Verlag, 2000.

[20] G. Rudolph, ''Convergence Analysis of Canonical Genetic Algorithms,'' *IEEE Trans. Neural Networks,* Vol. 5, No. 1, Jan. 1994, pp. 96-101.

[21] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17-26.

[22] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 25, 1997, pp. 8-22.

[23] M. Y. Wu, W. Shu, and H. Zhang, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," *9th IEEE Heterogeneous Computing Workshop* (*HCW 2000*), May 2000, pp. 375–385.

Appendix

Pseudocode for generating the DAGs

```
/* input:
   Na subtask nodes with no predecessors and only
     successors, with id #s ranging from 1 to Na
   Nb subtask nodes with both predecessors and
     successors, with id #s ranging from Na+1 to
     Na+Nb
   Nc subtask nodes with no successors and only
     predecessors, with id #s ranging from Na+Nb+1
     to Na+Nb+Nc
   maxFanOut, the maximum number of edges out of a
     node
   minFanOut, the minimum number of edges out of a
     node
*/
/* output:
   a DAG where all edges point from a smaller id node
     to a larger id node
*/
DAG generator pseudocode

1)  for every node with successors, i,
        /* the maximum number of outgoing edges of
           node i must be equal to the maximum
           fanout or the number of nodes with id larger
           than node i */
2)      maxedges = min(maxFanOut, number of
        nodes with id larger than i)
3)      generate a random number, j, between
        (minFanOut, maxedges)
4)      randomly select j nodes with id larger than i
        and generate an edge from i to each of the j
        nodes, updating the data structures
        accordingly
5)  endfor

/* check for nodes from (Na +1) to (Na+Nb+Nc) that
do not have an incoming edge*/
6.  for each node, i,
7.      if there is no incoming edge
          /* find the first node with id less than i that
          can be used to make an edge to the node i */
8.        for k =1 to (i –1) do
9.          if k does not have max outgoing edges
10.           generate an edge between the node k
              and the node i, and break out of this for
              loop
11.         else if k has an outgoing edge pointing to a
              node that has more than 1 incoming
              edge
12.           move the outgoing edge to point to
              node i, and break out of this for loop
13.         endif /* matches the if in Line (9) */
14.       endfor /* matches the for in Line (8) */
15.     endif /* matches the if in Line (7) */
16.  endfor /* matches the for in Line (6) */
```

End of DAG generator pseudocode.

Biographies

Sameer Shivle is a graduate student of Colorado State University pursuing his M.S. degree in Electrical and Computer Engineering. He received a B.E. degree in

Electrical Engineering from the Government College of Engineering, Pune, India. He has worked as a Software Engineer with Mahindra British Telecom, India from 2000 to 2001. His fields of interest are heterogeneous computing, computer architecture, resource management, and digital system design.

Ralph Castain is currently serving as a Research Scientist within the Electrical and Computer Engineering Department at Colorado State University where he conducts research focused on robust resource management within distributed computing systems. In addition, he is leading the Colorado Grid Computing (COGrid) initiative on behalf of Colorado State University, an effort that he founded in late 2002 to create a statewide grid computing system capable of meeting the needs of industry, government, and academia of all levels. This initiative is rapidly gathering momentum, with production capabilities expected to become operational in early 2004. Prior to joining the University, he spent eight years in industry leading technology initiatives, and eleven years at Los Alamos National Laboratory. While at Los Alamos, he served as Chief Scientist - Nonproliferation and Arms Control. His technical paper in the early 1990s on next-generation methods for proliferation detection has served as the foundation for the U.S. government's nonproliferation research program for over ten years. He received his BS degree from Harvey Mudd College, and the MS, MSEE, and PhD degrees from Purdue University.

H. J. Siegel holds the endowed chair position of Abell Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU), where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC). ISTeC a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering at Purdue University. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 300 technical papers. His research interests include heterogeneous parallel and distributed computing, communication networks, parallel algorithms, parallel machine interconnection networks, and reconfigurable parallel computer systems. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and has been on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of four international conferences, and Chair/Co-Chair of five workshops. He is currently on the Steering Committees of five continuing conferences/workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. *An up-to-date vita is available at www.engr.colostate.edu/~hj.*

Anthony A. Maciejewski received the B.S.E.E, M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University under the support of an NSF graduate fellowship. From 1985 to 1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan where he performed work on the development of parallel processing algorithms for computer graphic imaging. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University where he is currently the Head of the Department of Electrical and Computer Engineering. Prof. Maciejewski's primary research interests relate to the analysis, simulation, and control of robotic systems and he has co-authored over 100 published technical articles in these areas. He is an Associate Editor for the *IEEE Transactions on Robotics and Automation*, a Regional Editor for the journal *Intelligent Automation and Soft Computing*, and was co-guest editor for a special issue on "Kinematically Redundant Robots" for the *Journal of Intelligent and Robotic Systems*. He serves on the IEEE Administrative Committee for the Robotics and Automation Society and was the Program Co-Chair (1997) and Chair (2002) for the International Conference on Robotics and Automation, as well as serving as the Chair and on the Program Committee for numerous other conferences. *An up-to-date vita is available at www.engr.colostate.edu/~aam.*

Tarun Banka is currently pursuing Ph.D. in the Department of Computer Science at Colorado State University. He received his M.S. in Computer Science from Colorado State University and B.E. in Computer Science and Engineering from Punjab Engineering College, Chandigarh India. He has worked with Honeywell India as a Software Engineer from 1998 to

2000. His research interests are in the field of sensor networks and distributed computing.

Kiran Chindam is a graduate student of Colorado State University pursuing a M.S. degree in Electrical and Computer Engineering. He received his B.Tech. degree in Electronics and Communication Engineering from Jawaharlal Nehru Technological University (JNTU) in 2001. His research interests include VLSI systems design, heterogeneous, parallel and reconfigurable computing. He is a student member of IEEE.

Steve Dussinger received a B.S. in Electrical Engineering and a B.S. in Computer Engineering in 1999 from the University of Florida. He is currently pursuing a M.E in Electrical Engineering degree in the Department of Electrical and Computer Engineering at Colorado State University. He currently works for Hewlett Packard in Fort Collins as a VLSI design engineer. His technical interests include VLSI design techniques and computer architecture.

Prakash K. Pichumani graduated from the Department of Electrical and Computer Engineering at Colorado State University where he received his M.S. degree in Electrical and Computer Engineering. He received his Bachelor of Engineering degree in Electronics and Instrumentation Engineering from Annamalai University, India. His research interests include wireless networking, heterogeneous computing, digital communications, and digital signal processing.

Praveen Moses Satyasekaran is a graduate student in the Department of Electrical and Computer Engineering. He received his B.E. degree from Bharathiar University, Coimbatore, India. His research interests include exploring chip level parallelism, VLSI design, analog IC design, and mixed signal design.

William W. Saylor, PE, is currently working as consultant for the Department of Defense on several advanced technology programs and is also in a graduate program at Colorado State University doing research on control issues for complex systems. He has spent the past eight years working in the defense and energy industries after twelve years at the Los Alamos National Laboratory, where he was a project leader for several aerospace and defense efforts. Prior to that, he worked as a nuclear engineer in the energy industry and spent nine years in the U. S. Army. He received his B.S. degree from The United States Military Academy and an M.S. degree from MIT, and is a registered professional engineer in Pennsylvania and Colorado.

David Sendek is pursuing a Ph.D. from the Department of Electrical and Computer Engineering at Colorado State University. His main research interest is in VLSI computer architectures. He received his B.S. in Mathematics from the College of Charleston in 1981 and his M.S.E.E. from the Naval Postgraduate School in 1990. Prior to pursuing higher education, Commander Sendek served in the United States Navy as an engineer where he managed the acquisition, research and development, systems engineering, and life cycle support of military satellite communications systems and missile weapons systems. During active duty military service, he obtained advanced level Defense Acquisition Workforce Improvement Act certifications in Program Management, Systems Engineering, and Test and Evaluation Engineering.

JC Sousa is pursuing a M.E in Electrical Engineering degree in the Department of Electrical and Computer Engineering at Colorado State University. He received a B.S. degree in Electrical Engineering from University of Utah. He is currently employed at Hewlett Packard and is working on microprocessor design for servers.

Jayashree Sridharan is a graduate student of Colorado State University pursuing an M.S. degree in Electrical and Computer Engineering. She received her Bachelors degree in Electronics and Communication Engineering from M. S. Ramaiah Institute of Technology (affiliated to the Visveswaraiah Technological University, Karnataka) in 2002. Her research interests include computer architecture, system level hardware design, hardware description languages, and VLSI.

Prasanna V. Sugavanam is pursuing his M.S. degree in Electrical and Computer Engineering at Colorado State University, where he is currently a Graduate Teaching Assistant. He received his Bachelor of Engineering degree in Electrical and Electronics at Bharathiar University, India. His current focus includes heterogeneous computing, computer architecture, re-configurable and microprocessor based systems. He is a student member of IEEE Computer Society.

José Velazco received a B.S. degree in Electrical Engineering from the University of Puerto Rico at Mayagüez. He then received a M.E. degree from Colorado State University. He has worked in the semiconductor industry with Hewlett-Packard in Fort Collins, CO, and is currently employed by

Abbott Laboratories in a manufacturing facility in Barceloneta, PR. His research interests include VLSI design, analog IC design, networking, and digital communications.